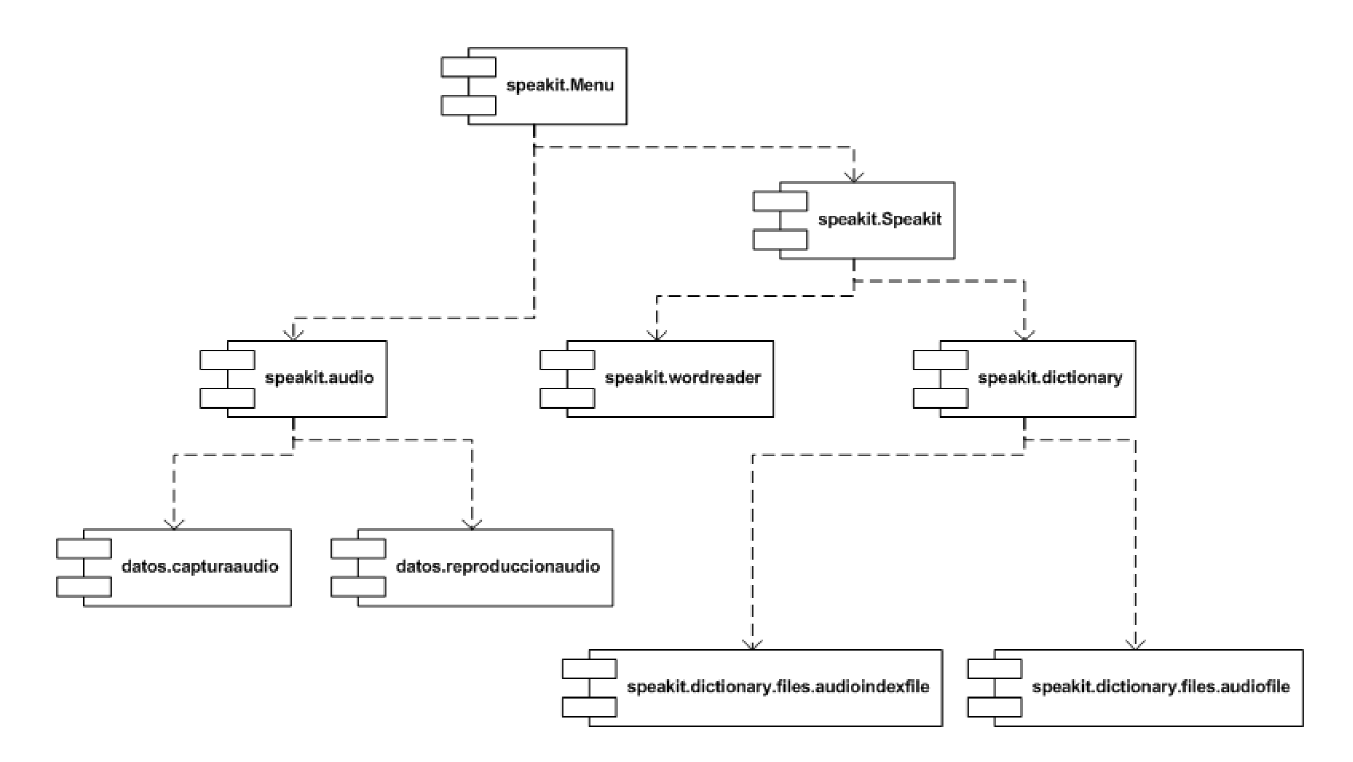
SPEAKIT!

DOCUMENTACIÓN TÉCNICA

Arquitectura del sistema (primer etapa).

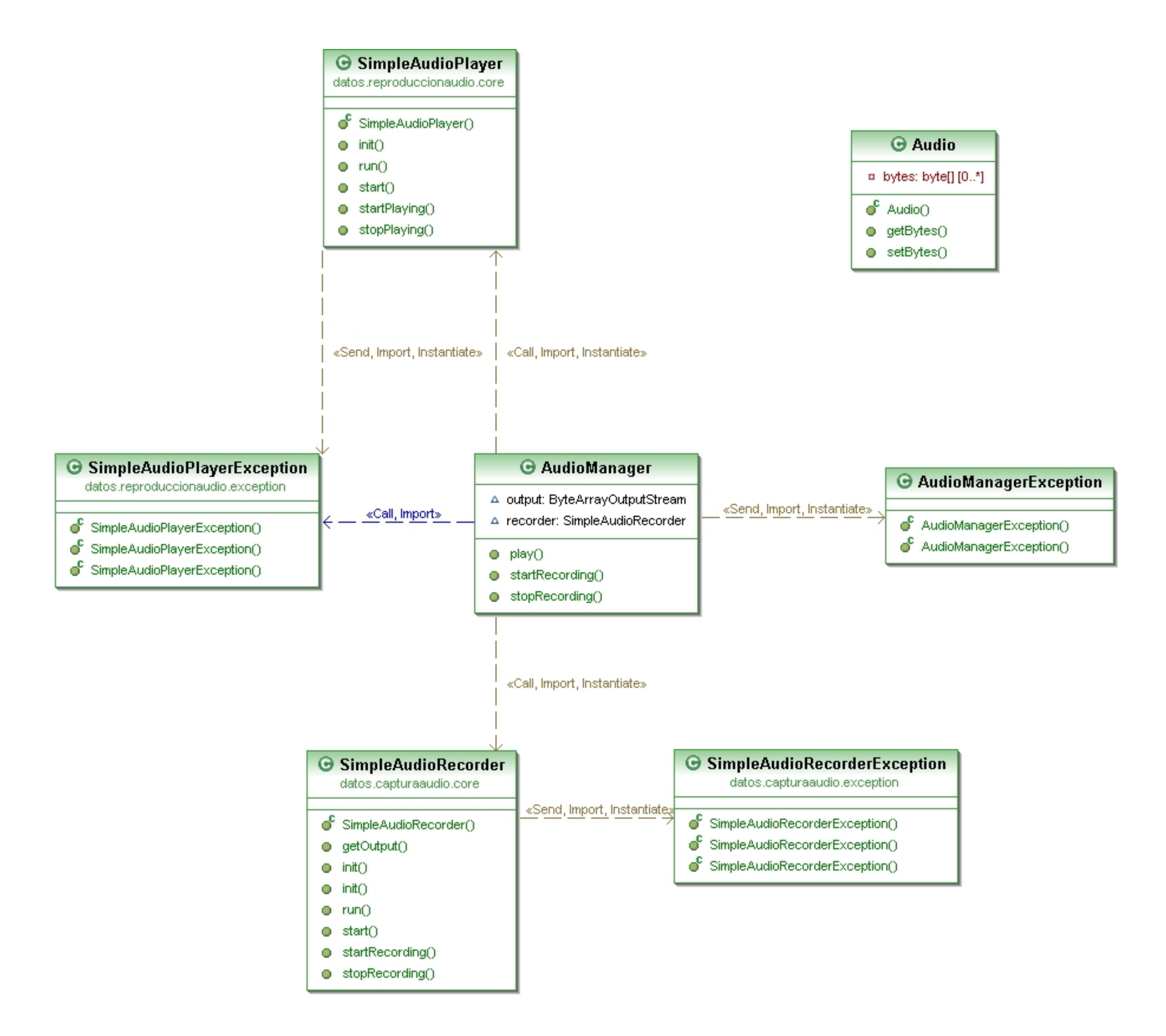
El sistema consta de 5 módulos bien diferenciados, a saber:
-Un módulo de audio.
-Un módulo lector de palabras.
-Un módulo de acceso a datos.
-Un módulo de modelo de negocio.
-Un módulo de interfaz de usuario.

Diagrama que ilustra las dependencias entre componentes.



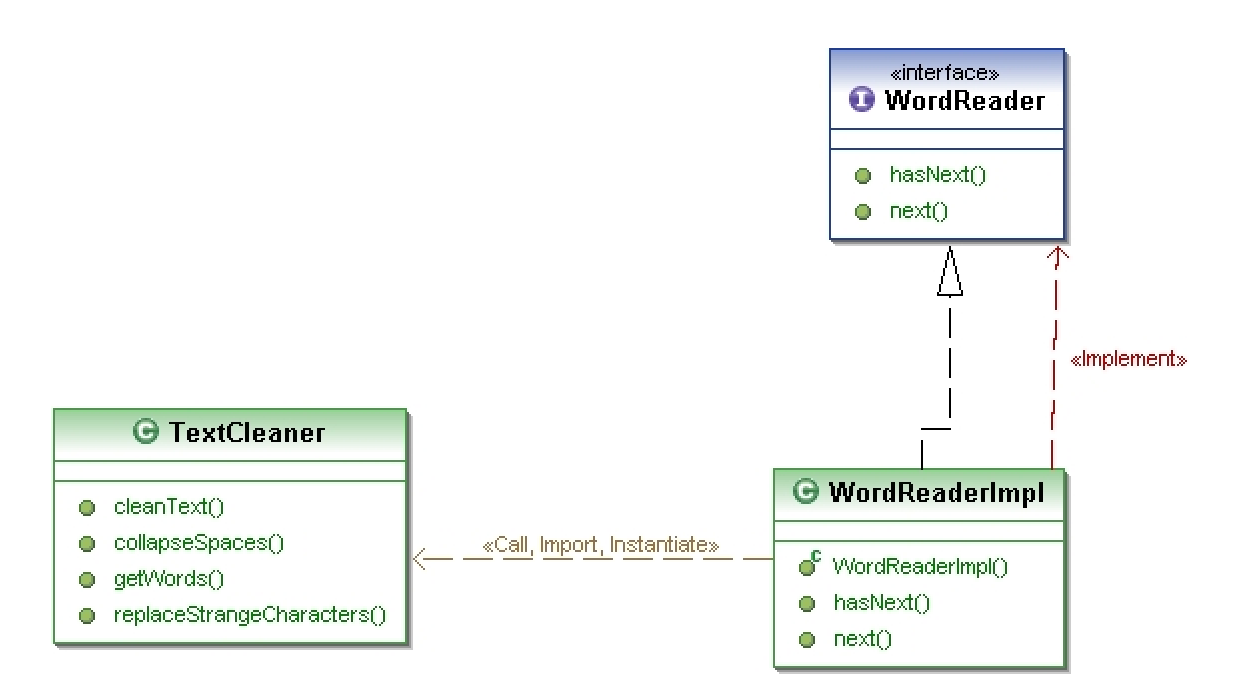
speakit.audio

El módulo de audio tiene la funcionalidad de grabar o de reproducir audio. Está implementado en el paquete speakit.audio, y consiste en una clase proxy que encapsula las funcionalidades del paquete de audio provisto por la cátedra. Tiene funciones para reproducir un objeto Audio, o grabar un objeto Audio.



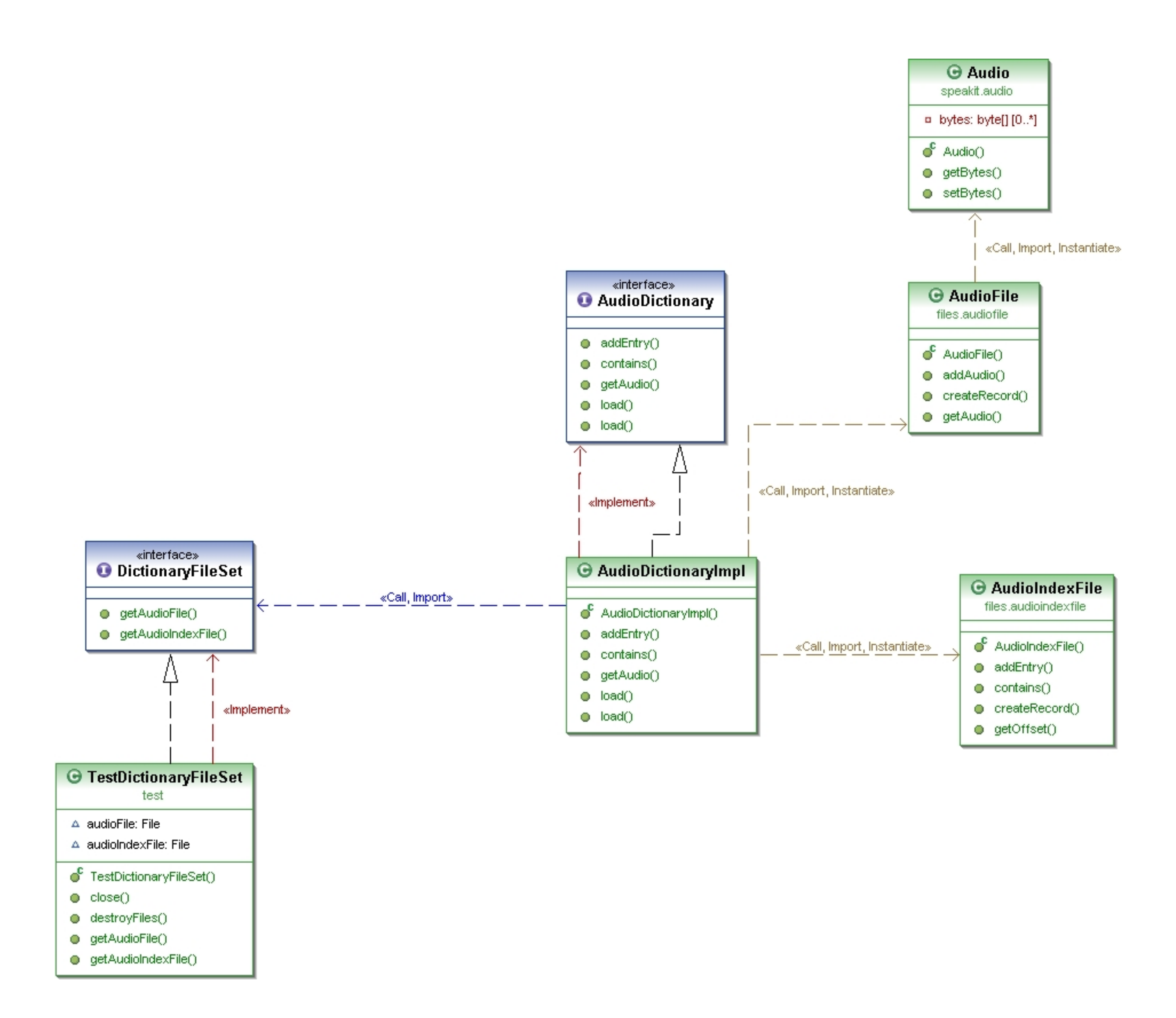
speakit.wordreader

El módulo lector de palabras (paquete speakit.wordreader), lee las palabras de un documento de texto. Para poder hacer esto, realiza tareas de limpieza, que eliminan caracteres de puntuación, y otros caracteres no latinos, colapsa los espacios, y convierte en minúsculas. Luego de ese tratamiento, devuelve un iterador de palabras.



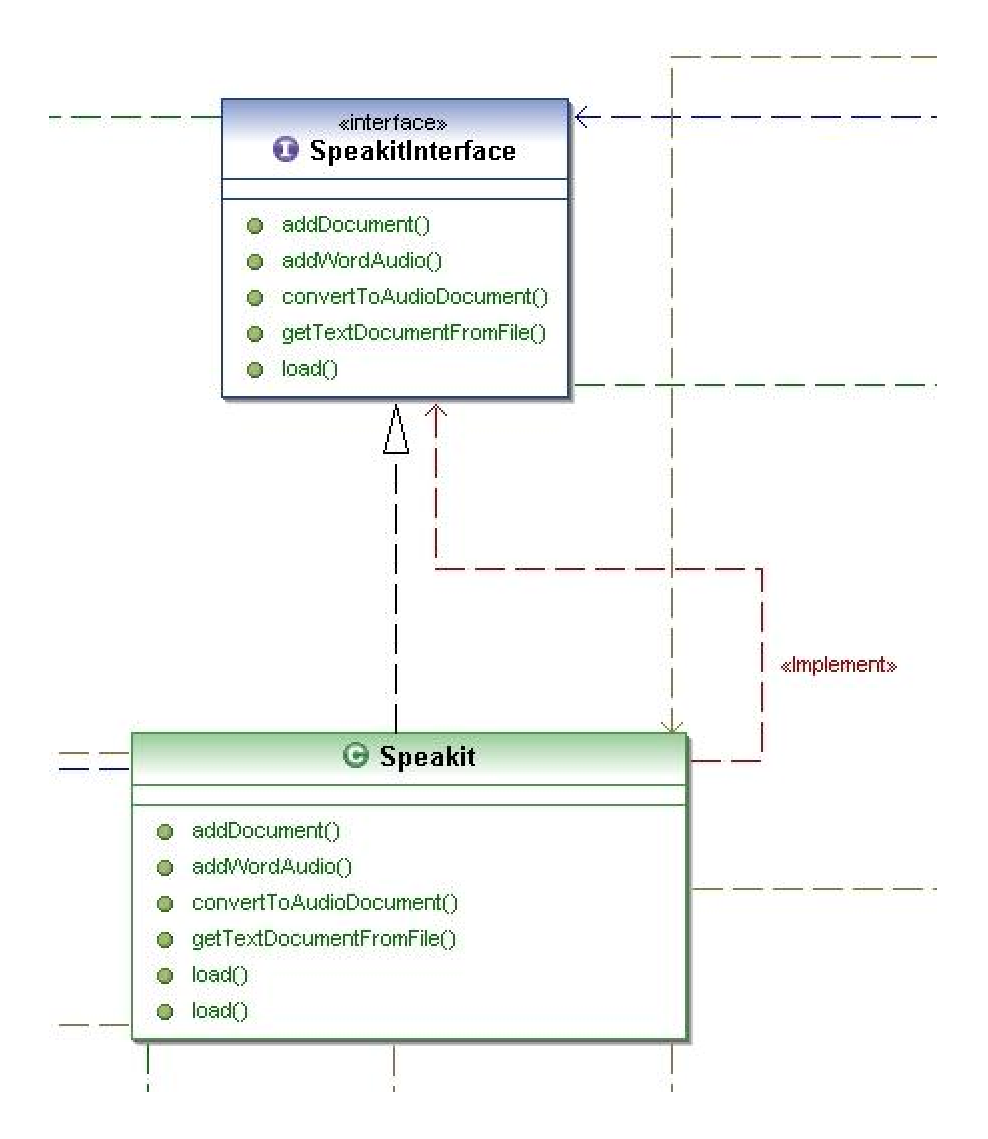
speakit.dictionary

El módulo de acceso a datos, contiene primitivas para poder obtener el audio de palabras grabadas en los archivos binarios del sistema. Este módulo viene implementado en el paquete speakit.dictionary. Contiene 2 submódulos. Un módulo para el acceso al archivo de audio y un módulo para el acceso al archivo que indiza el archivo de audio.



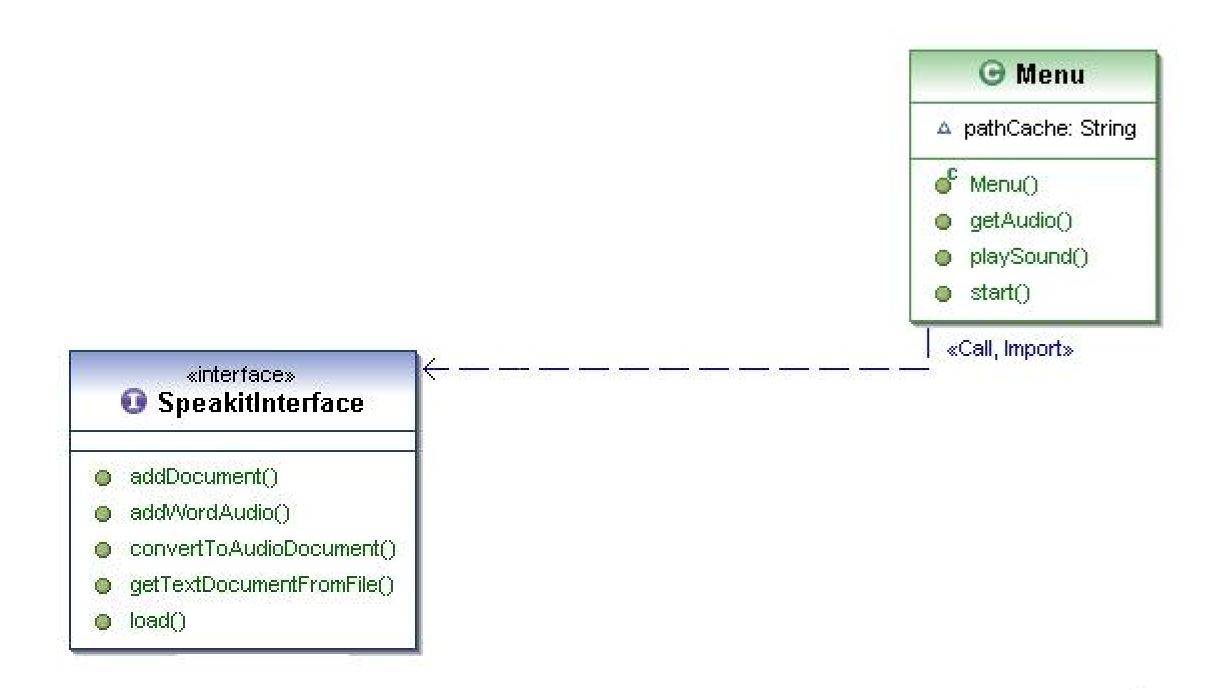
speakit.Speakit

El módulo del modelo de negocio, simbolizado por la clase Speakit, integra los demás módulos. Tiene la función de agregar un documento (función pensada para la siguiente etapa, que ahora se limita a devolver las palabras desconocidas), agregar palabra, y convertir un documento de texto en un documento de audio.



speakit.Menu

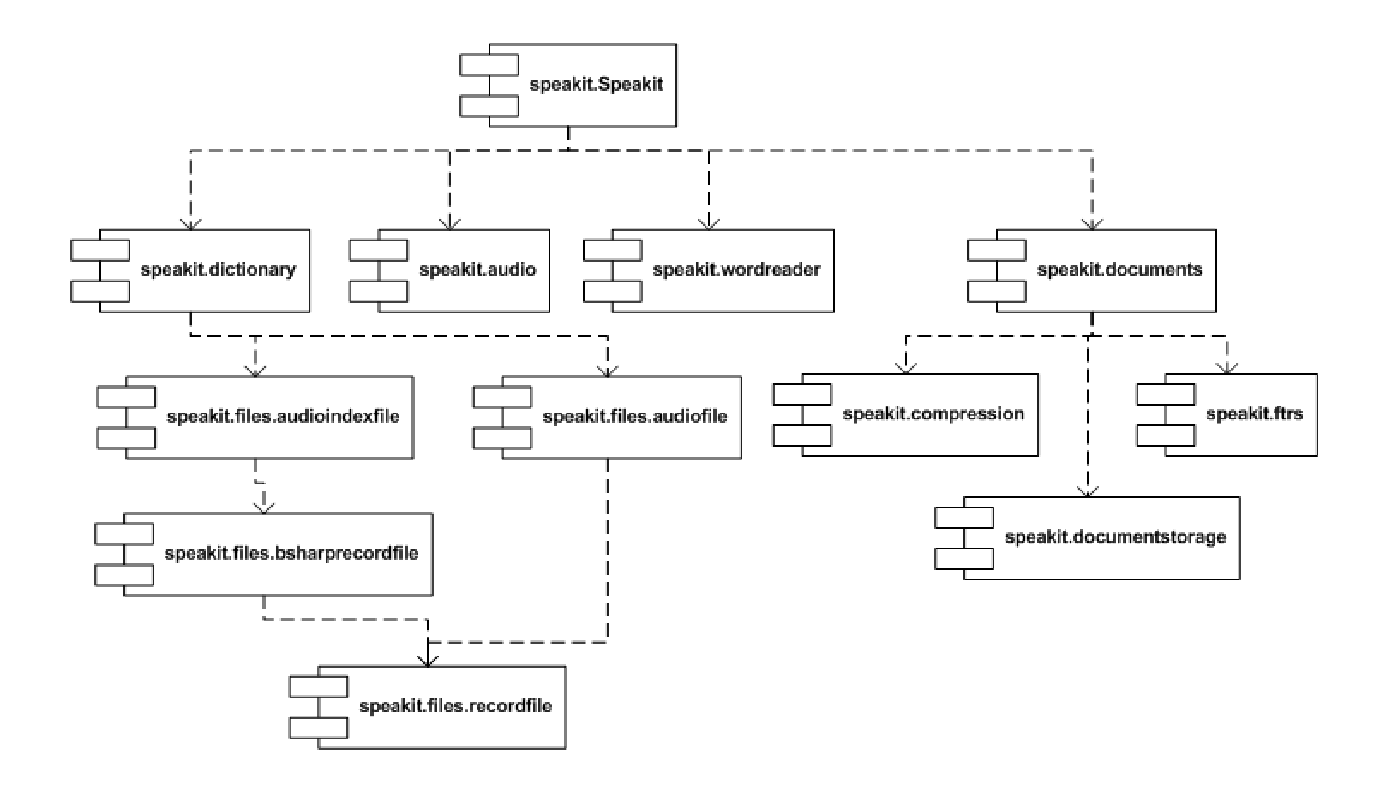
El módulo de interfaz de usuario es el menú, y el que gestiona el control del programa, el que le da instrucciones a Speakit para obtener o guardar una palabra, muestra textos en pantalla, permite la entrada del usuario, y utiliza al módulo de audio para grabar y reproducir sonido.



Arquitectura del sistema (otras etapas)

El sistema Speakit, actualmente permite ingresar un documento de texto, grabar todas las palabras que no estén registradas, y agregarlas a un diccionario que asocia palabras con su audio, y también permite reproducir las palabras contenidas en un documento.

Speakit será expandido en la siguiente etapa para agregar la funcionalidad de guardar los documentos y poder buscarlos. En una etapa posterior estos documentos y los audios se guardarán en forma comprimida.



El módulo Speakit, conservará la función de agregar el audio de una palabra. Además tendrá la función de agregar un documento. Esta función realizará su tarea valiéndose del módulo documents. Podrá almacenar documentos a pesar de que el módulo dictionary no contenga todas sus palabras. También tendrá la función de obtener todas las palabras desconocidas de los documentos almacenados, lo cual servirá para grabar nuevas palabras.

Los módulos *speakit.audio*, *speakit.wordreader* y *speakit.dictionary* se mantendrán. Se agregará un módulo *speakit.documents*. Los demás módulos se modificarán de la siguiente forma:

dictionary no cambiará, mientras que los módulos de los cuales depende serán cambiados. Por ejemplo, audiofile dejará de utilizar el módulo recordfile actual (de implementación secuencial), y utilizará uno nuevo, que implemente la organización en bloques. Esto es conveniente para permitir la modificación de los registros de audio, sin necesidad de reestructurar el archivo. *audioindexfile* cambiará para estar implementado con un árbol B#. El árbol B# será otro módulo, que a su vez utilizará el mismo módulo *recordfile* que utiliza *audiofile* para implementar su persistencia.

El módulo *documents* tendrá la funcionalidad de guardar, y buscar los documentos de texto guardados. Los documentos agregados, serán comprimidos con el módulo compression, y serán guardados luego con el módulo *documentsstorage*. La búsqueda de textos se implementará en el módulo *ftrs*, y una vez encontrado el documento, será recuperado mediante su clave del módulo *documentsstorage*.

Organización de Archivos

El sistema maneja dos archivos de datos, el archivo de audio, y el archivo de índice.

El archivo de audio (AudioFile), es un archivo de datos maestro, aquí se guardan físicamente todos los sonidos grabados por el usuario en forma de registros.
El archivo de índice (AudioIndexFile), registra la asociación entre palabras y registros del archivo de audio.
En ambos ficheros, los registros son de longitud variable y quedan almacenados uno a continuación del otro, es decir, no existen estructuras de bloques.

Definición Conceptual de Datos:

AudioFile: AudioRecord((offset)i,audio)
AudioIndexFile: AudioIndexRecord((word)i,(offset)ie)

Referencias:
i = identificador
ie = identificador externo

Aclaración: el campo offset del AudioRecord, en realidad no existe fisicamente en cada registro, pero como es la forma de identificarlo, lo quisimos expresar en la definición conceptual.

Definición Lógica de Datos:

AudioFile: AudioRecord(longAudio: integer, audio: byte array)
AudioIndexFile: AudioIndexRecord(offset: long, longWord: integer, word: string)

Soluciones descartadas

Grabación y reproducción

El proyecto requiere que reproduzcamos el audio grabado, para esto la cátedra nos proveyó de un módulo que realiza esta tarea. Nosotros debíamos hacer que las palabras se reprodujeran una a continuacion de la otra, en serie, pero el módulo de audio estaba diseñado como un thread que se iniciaba cuando se le enviaba un sonido y moría cuando terminaba de reproducir, lo que nos indicaba que debíamos hacer alguna operación entre threads para que el hilo principal se detuviera hasta finalizar cada palabra.
Intentamos hacer operaciones entre threads y no lo logramos. Llegamos a la conclusión de que el thread de reproducción nunca finalizaba. Entonces optamos por una solución casera, pero no menos ingeniosa, de modificar la sección del programa que se encargaba de la grabación para que, ademas de grabar audio, registrara el tiempo durante el cual el usuario había estado hablando. Esto lo almacenábamos como un campo más dentro de cada registro del archivo de sonidos. Luego al reproducir utilizabamos esa información para dormir al hilo principal mientras el de reproducción corría.
Esta solución funcionó muy bien, pero el manejo de concurrencia no era muy elegante.
Unos dias después, la cátedra envió una nueva versión de la librería de sonido, e hicimos otro intento por manejar la concurrencia de forma apropiada, modificamos el código para

que el thread principal invocara al método *join* del objeto que reproducía, esto funcionó correctamente, el hilo principal esperó a que el hilo de sonido muriera, tal como queríamos. Esta última solución fué mucho mas simple que la primera, por lo que deshicimos todo lo anterior y lo redujimos a una sola linea de código.

Palabras no encontradas

Para obtener las palabras no reconocidas de un documento de texto se nos ocurrió definir una hipótesis: *Sólo son válidos dentro del sistema los documentos cuyas palabras tengan su representación sonora registrada*. Entonces, cuando agregabamos un documento con palabras nuevas, el documento no cumplía con la hipótesis por lo que el sistema arrojaba una excepción indicando la palabra que era desconocida, el menú la captaba y le pedía al usuario que la grabe, luego el programa principal reintentaba ingresar el documento, el sistema volvía a arrojar una excepción. Así sucesivamente hasta que todas las palabras estén grabadas. Esto funcionaba pero tenía algunos problemas.

- *Conceptual*: Un documento con palabras nuevas no es un caso excepcional, es una funcionalidad del sistema, por lo que no es correcto arrojar una excepción por eso.
- *Performance*: Cada vez que intentabamos agregar un documento, el sistema recorría todas las palabras del mismo en busca de alguna desconocida. Esto requería que cada intento de agregarlo leíamos archivo, lo correcto sería hacer una sola pasada.
- *Mal uso de excepciones*: Utilizabamos las excepciones para controlar el flujo del programa, esto es una mala práctica.
- *Funcional*: No debería ser un inválido que algunas palabras de un documento no estén registradas, en todo caso, a la hora de reproducirlo no se escucharían y se continuaría con la siguiente. La hipótesis era demasiado restrictiva.

Finalmente nos decidimos por eliminar esta hipótesis y devolver las palabras desconocidas en el mismo método que agrega un documento. Si bien al usuario inmediatamente se le pide que grabe las palabras, el documento ya queda en el sistema. Si la interfaz lo permitiera se podría abortar la grabación de palabras dejando así un documento, con algunas palabras desconocidas, en un estado totalmente consistente.

Separación vista - modelo

Queríamos independizar lo maximo posible el modelo de la vista y pensamos tener una clase que se maneje el flujo del programa y la logica de negocio, y otra de la vista. La que se encargaba del flujo del programa, llamada speakit, exponía sólo dos primitivas; una para agregar y otra para reproducir un documento. Para comunicarse con el usuario pensamos en un patrón observador. La clase main creaba a speakit y le pasaba como observador al menú, este respondería a eventos que le indicaba si debia mostrar el menu de presentacion, solicitar que el usuario grabe un audio, reproducir un sonido, etc...
Esta solución era extremadamente compleja de programar (desarrollamos solo una parte), y hacía inentendible el código, pues el modelo debía encargarse del flujo del programa y ademas de comunicarse con los módulos de datos.
Descartamos esta solucion e hicimos que la clase que se encargue del flujo del programa y la que se comunique con el usuario sea la misma y que el modelo sólo maneje a los módulos de datos. Eliminamos el patrón observador y dejamos al modelo como una clase

que provee servicios tales como agregar un documento de textos devolviendo las palabras desconocidas, y obtener un documento de audio a partir de uno de textos.

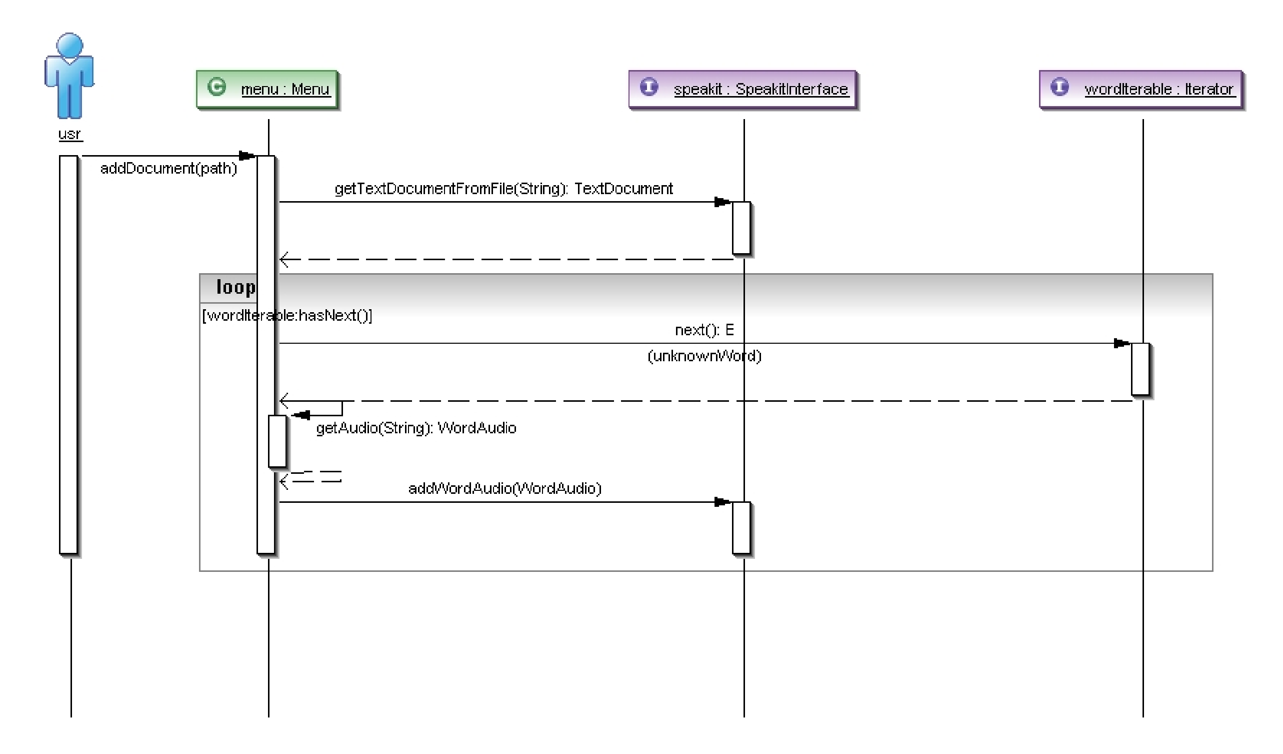
Diagramas de Secuencia de la aplicación

Agregar un nuevo documento

Escenario: “Se agrega un documento de texto, el cual contiene algunas palabras desconocidas.”

Secuencia resultante:

El usuario envía el mensaje “addDocument” al menú incluyendo path del documento a procesar.
El menú envia el mensaje “getTextDocumentFromFile” con el path del documento como parametro a speakit, recibiendo como respuesta un TextDocument , wordIterable, incluyendo las palabras desconocidas.
Se inicia un loop, q se ejecuta mientras wordIterable contenga palabras.
El menú le pide a wordIterable la siguiente palabra desconocida
El menú se envia un auto mensaje “getAudio()” pasandose la palabra desconocida como parametro, obteniendo como respuesta wordAudio, que corresponde al audio de la palabra consultada.
El menú envía el mensaje “addWordAudio” a speakit pasandole a wordAudio como parametro
Finaliza el loop.
Finaliza la secuencia.



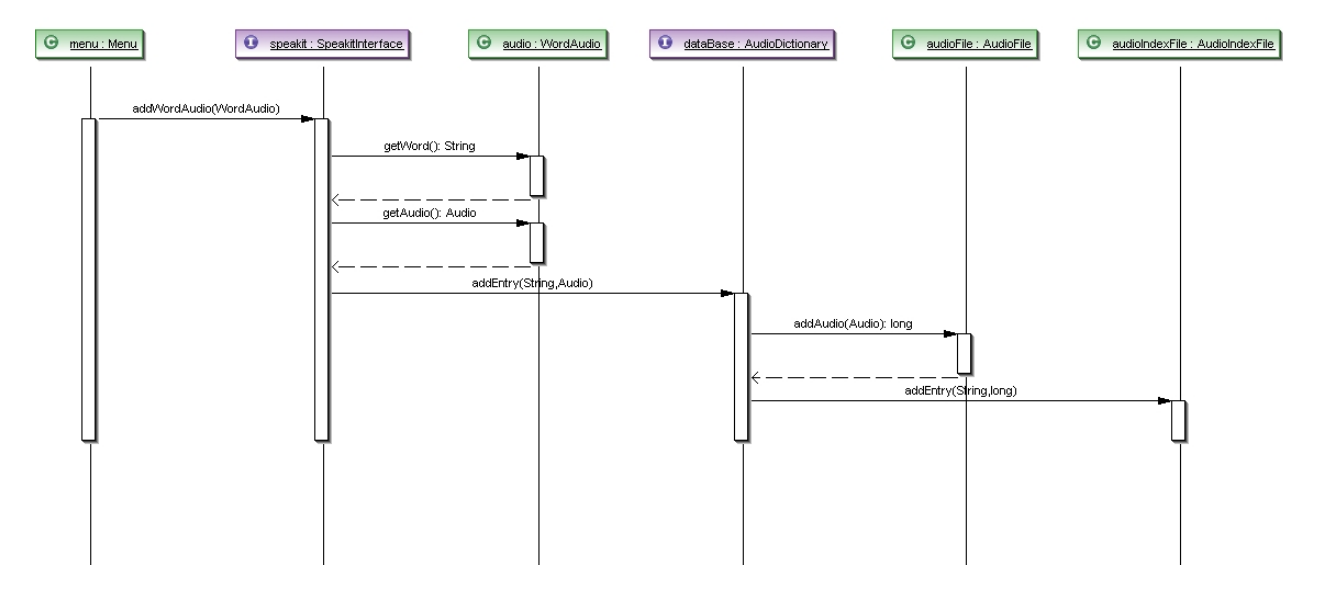
Almacenar una nueva palabra

Escenario: “Almacenando una nueva palabra en el sistema”

Secuencia resultante:

El menú envía a speakit el mensaje “addWordAudio” con la nueva palabra registrada como parámetro (audio).
Speakit envía el mensaje “getWord” a audio, recibiendo como respuesta el String q representa a la palabra en texto.
Speakit envía el mensaje “getAudio” a audio, recibiendo como respuesta una cadena de bytes q representa a la palabra en formato audio.
Speakit envía el mensaje “addEntry” a database pasándole como parámetros el string y la cadena de bytes.
Database envía el mensaje “addAudio” a audioFile con la cadena de bytes como parámetro, recibiendo como respuesta el offset a la posición correspondiente a la palabra recién agregada.
Database envía el mensaje “addEntry” a audioIndexFile, pasándole como parámetros el string representante de la palabra y el offset q le corresponde.

Fin de la secuencia.



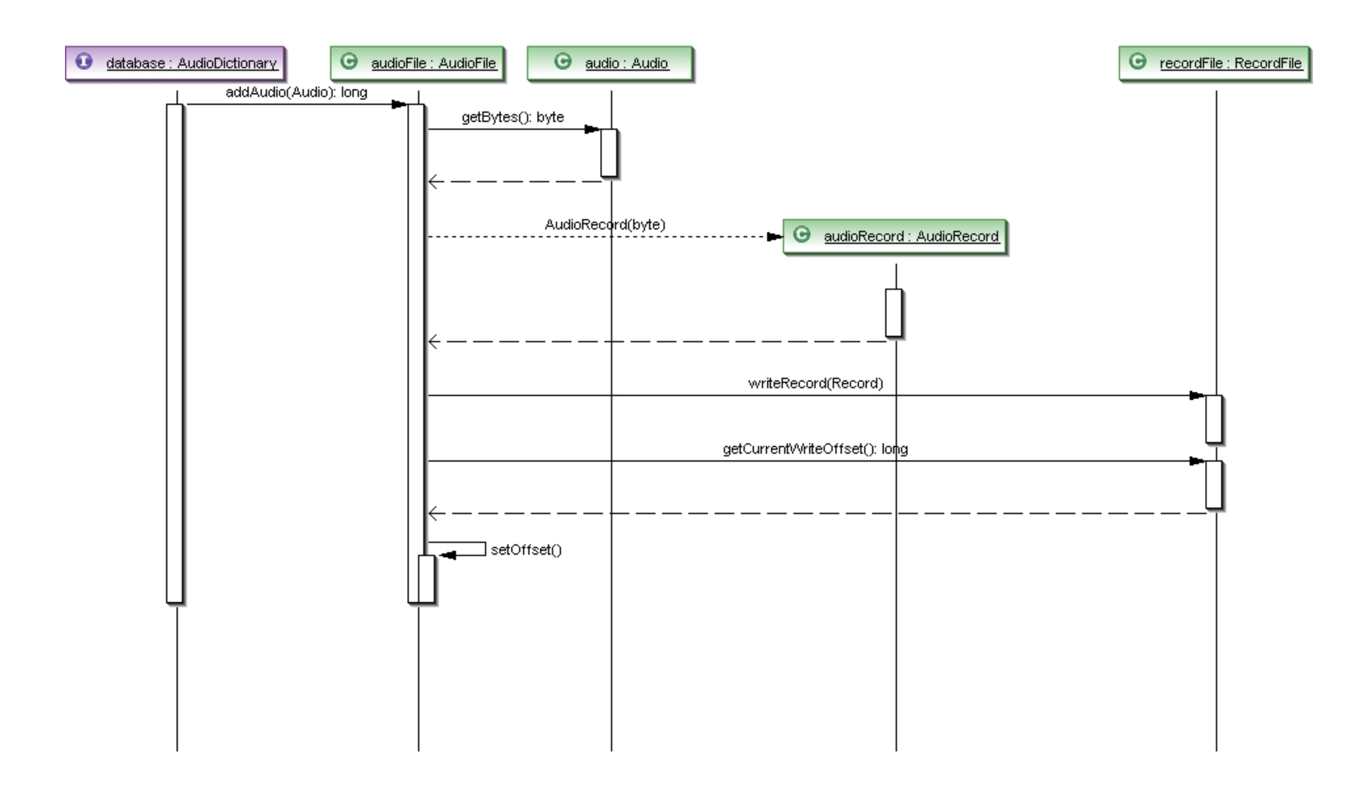
Agregar entrada al repositorio

Escenario: “El AudioDictionary agrega una nueva entrada al repositorio de palabras almacenadas en formato audio.”

Secuencia resultante:

El AudioDictionary envía el mensaje “addAudio” al audioFile, pasándole como parámetro el audio a grabar.

El audioFile obtiene la cadena de bytes que representa al audio mediante el mensaje “getBytes”
El audioFile crea una nueva instancia de AudioRecord, pasándole por parámetro la cadena de bytes anteriormente obtenida del objeto audio.
El audioFile envía el mensaje “writeRecord” al recordFile, pasándole como parámetro la nueva instancia de AudioRecord creada.
El audioFile envía el mensaje “getCurrentWriteOffset” al recordFile, obteniendo como respuesta la posición del registro en el archivo de registros.
El audioFile actualiza el valor del offset.
Fin de la secuencia.



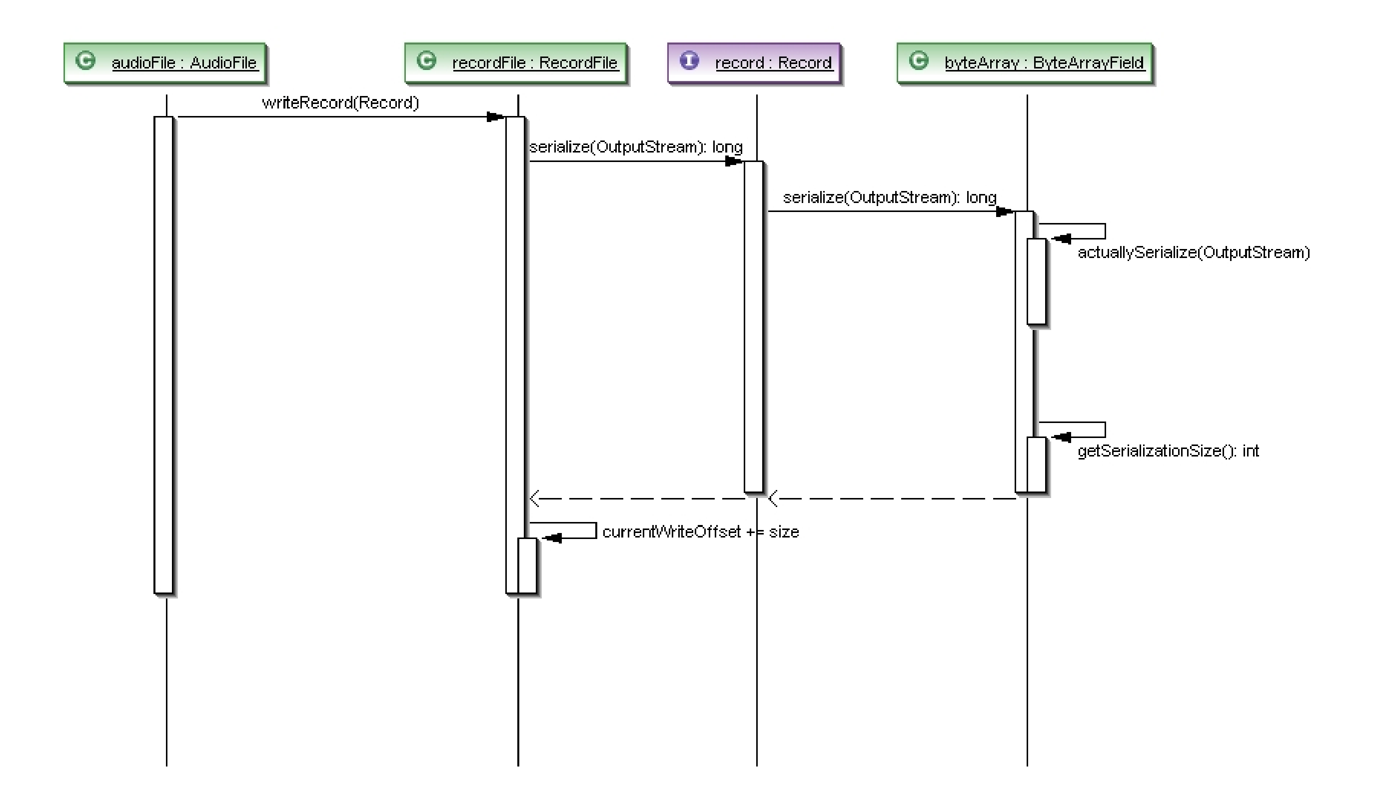
Persistir un registro

Escenario:”Una instancia de RecordFile recibe el mensaje “writeRecord” con el registro de tipo AudioRecord a escribir como parámetro y se encarga de persistirlo en el archivo de registros correspondiente”.

Secuencia resultante:

El audioFile envía el mensaje “writeRecord” a recordFile, pasándole como parámetro el registro a escribir.
RecordFile envía el mensaje “serialize” al record, pasándole como parámetro el OutputStream donde se escriben físicamente los registros.
El record deriva el mensaje “serialize” al byteArrayField que representa la porción de audio que contiene.

El byteArrayField se envía el mensaje “actuallySerialize”, encargado de la serialización propiamente dicha del registro.
El byteArrayField obtiene de sí mismo el tamaño insumido en la serialización del registro, a través del mensaje “getSerializationSize” enviándolo como valor de retorno al mensaje serialize enviado anteriormente por el record.
El record envía el valor recibido del byteArray como respuesta al mensaje serialize enviado por el recordFile.
El recordFile actualiza el valor del currentWriteOffset.

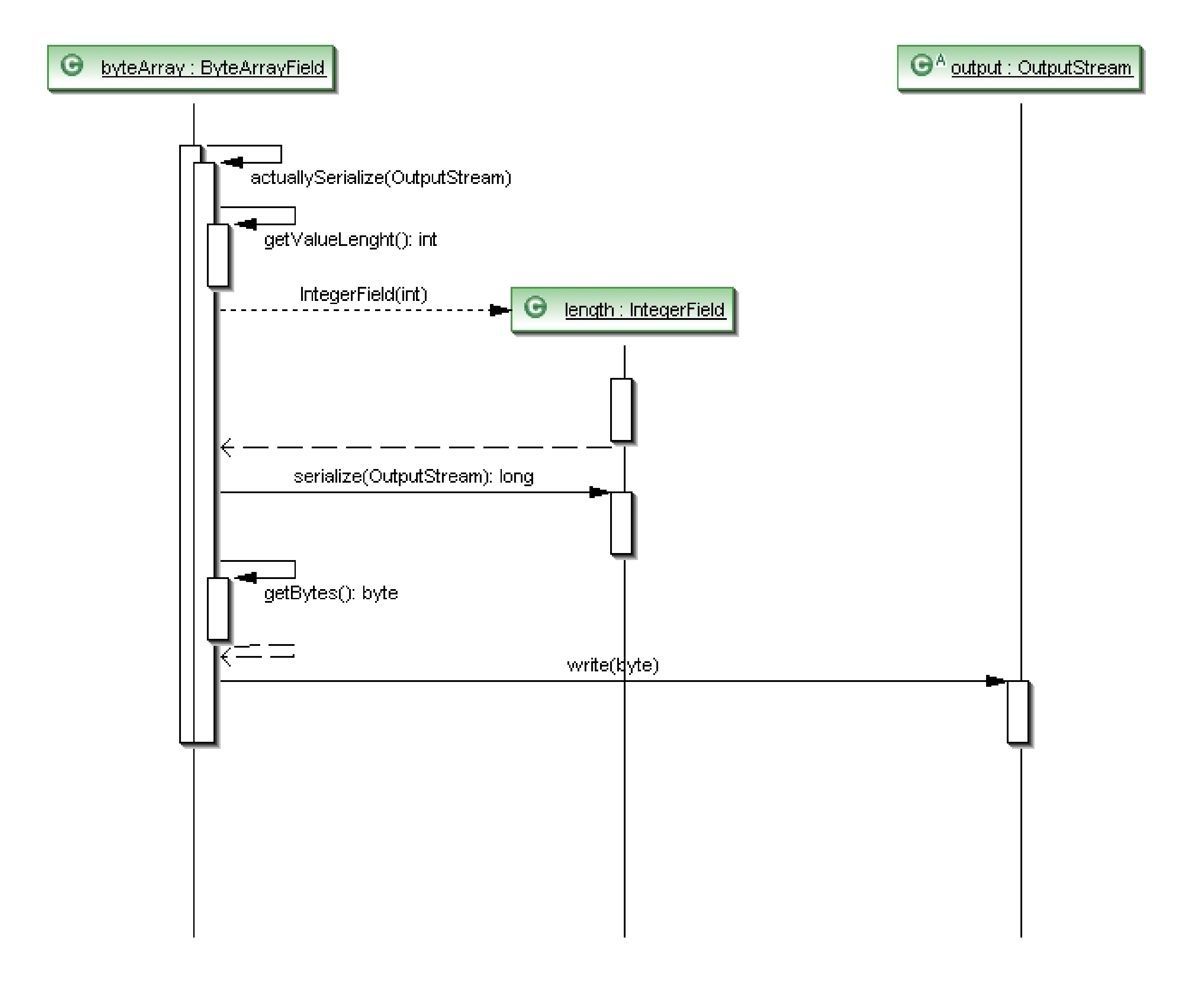


Serializacion de un campo

Escenario:”Serialización de un campo del tipo ByteFieldArray”

Secuencia resultante:

El byteArrayField se envía a sí mismo el mensaje “actuallySerialize”, pasándose como parámetro el outputStream donde escribir.
El byteArrayField obtiene el tamaño a escribir a través del mensaje “getValueLength”
El byteArrayField crea una nueva instancia de la clase IntegerField, pasándole como parámetro el tamaño a escribir.
El byteArrayField envía el mensaje “serialize” al integerField, pasándole como parámetro el outputStream donde escribir.
El byteArrayField se envía el mensaje “getBytes”, obteniendo así la cadena de bytes a escribir.
El byteArrayField le envía al outpuStream el mensaje “write” con la cadena de bytes como parámetro.



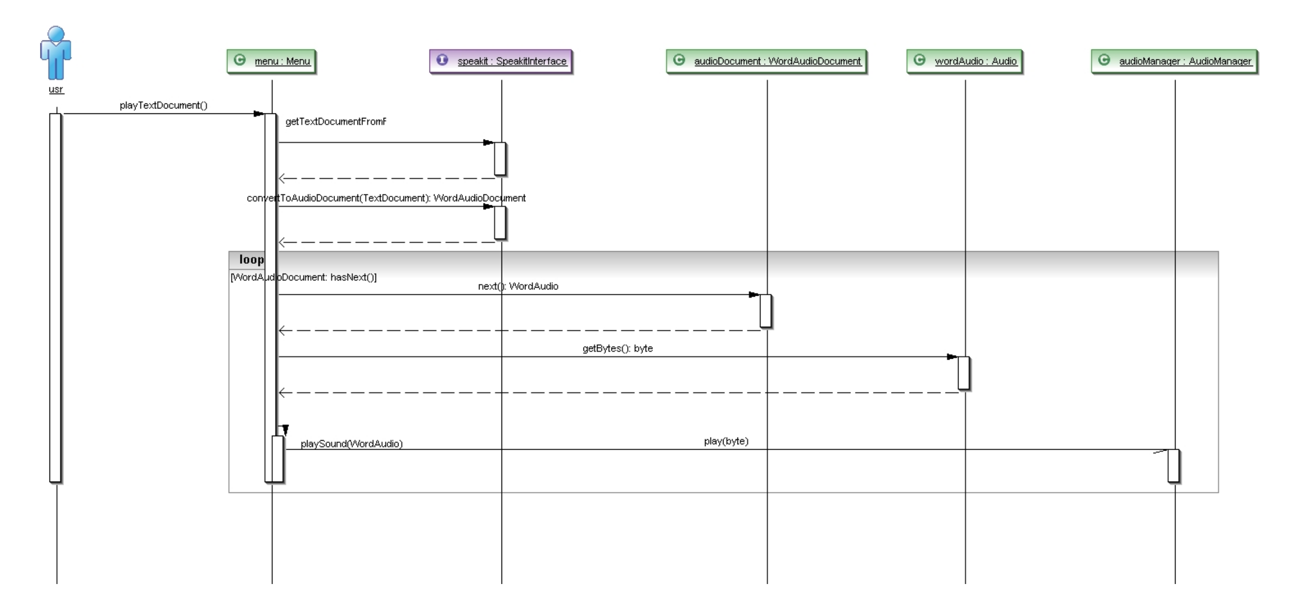
Reproducción de un archivo

Escenario:"El usuario solicita la reproduccion de un archivo de texto que se encuentra previamente procesado por la aplicación"

Secuencia resultante:

El usuario, a través de la interface de la aplicación, envía el mensaje "playTextDocument" al menú.
El menu envía el mensaje "getTextDocumentFromFile" a speakit, pasandole como parametro el path del archivo a reproducir. Speakit devuelve una instancia de TextDocument q representa al archivo solicitado.

El menu envia a speakit el mensaje “convertToAudioDocument”, pasandole como parametro el textDocument, y recibiendo como respuesta una representacion del archivo que contiene el audio equivalente a cada palabra
Se inicia un loop que se ejecuta mientras wordAudioDocument contenga palabras.
El menu obtiene la siguiente palabra del wordAudioDocument.
El menú obtiene la cadena de bytes correspondiente al audio mediante el mensaje “getBytes”
El menú se envia el mensaje “playSound” pasandose como parametro la cadena de bytes.
Finaliza el loop.
Finaliza la secuencia.

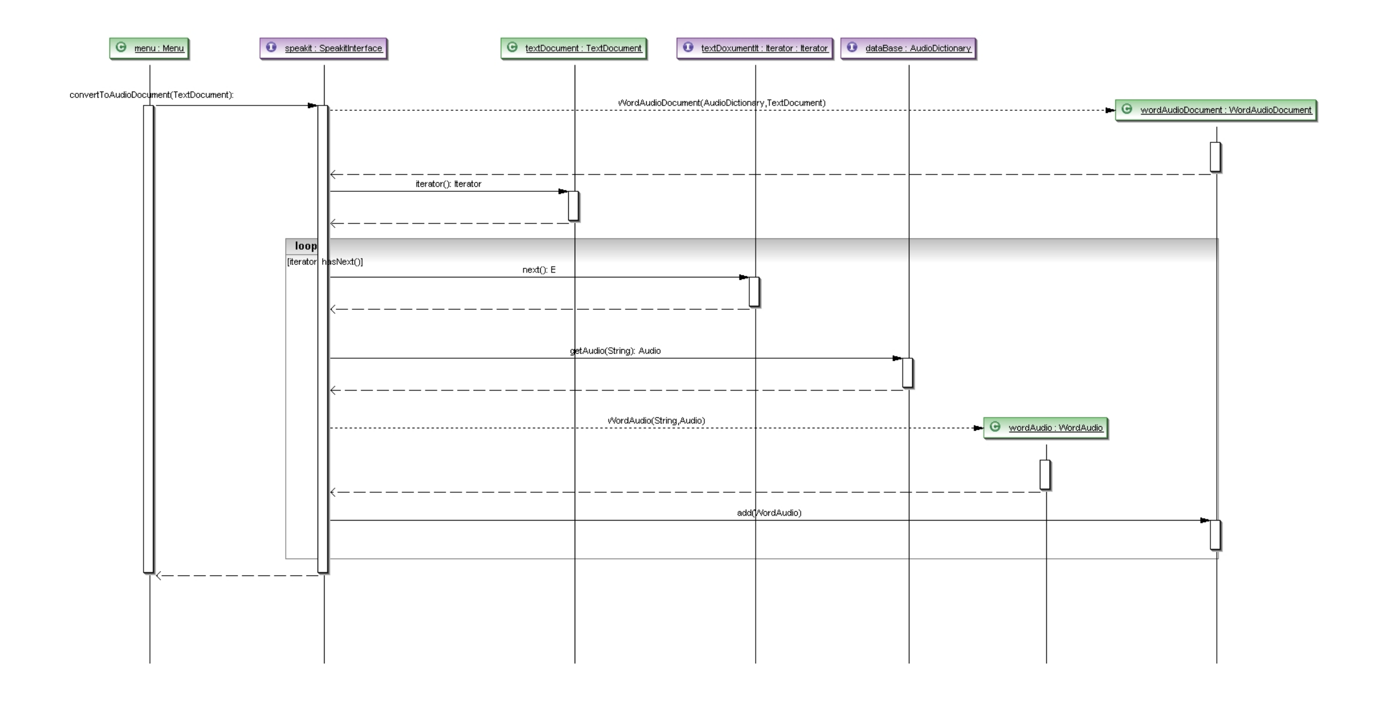


Generación de un archivo a reproducir

Escenario: “Se solicita la reproduccion de un documento, representado como un TextDocument, con todas las palabras registradas en el sistema.”

Secuencia resultante:

El menú envia el mensaje “convertToAudioDocument” a speakit, con el textDocument a recuperar.
Speakit crea una nueva instancia de la clase WordAudioDocument.
Speakit obtiene un iterador del textDocument.
Se inicia un loop que se ejecuta mientras textDocument tenga palabras.
Speakit obtiene un string representando a la siguiente palabra del textDocument
Speakit envia el mensaje “getAudio” a la instancia de AudioDictionary, obteniendo como respuesta la instancia de audio correspondiente.
Speakit crea una nueva instancia de wordAudio pasandole como parametro el string y el audio que representan a la palabra.
Speakit agrega la instancia de wordAudio al wordAudioDocument.
Finaliza el loop.
Speakit devuelve la instancia de wordAudioDocument como respuesta al mensaje enviado por el menú.



MANUAL DE USUARIO

Instrucciones de uso

Instalación del sistema

Para instalar y correr el sistema se requiere de la herramienta **Apache Ant**. Esta herramienta puede conseguirse mediante el gestor de paquetes de su distribución de Linux o descargandolo de la página **http://ant.apache.org** . Si se elige esta última opción se deben seguir las instrucciones de instalación de la herramienta proporcionadas en la página.

Para compilar o correr la aplicación, utilizando la consola del sistema operativo, ingresar al directorio **speakit** y ejecutar alguno de los siguientes comandos:

ant clean - Usando este comando se borran las carpetas utilizadas para compilar y distribuir la aplicación.

ant compile - Con este comando se compila la aplicación, generando los archivos .class dentro de la carpeta **build** del proyecto.

ant run - Al ejecutar este comando se compila y se ejecuta la aplicación.

ant - Si se ejecuta el comando ant, sin utilizar ningún parametro, por default se ejecuta el comando **ant run**.

Utilización del sistema

Al iniciar la aplicación podemos ver la siguiente pantalla:

```
root@osiris-desktop:/home/osiris/workspace/speakit# ant run
Buildfile: build.xml

init:

compile:

jar:

run:
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
```

Aqui tenemos 3 opciones:

- **Procesar archivo de Texto**: En este módulo el sistema pregunta por un archivo de texto para ser leido. Si el archivo está dentro de la carpeta de la aplicación no es necesario ingresar la ruta al mismo, solo se debe ingresar el nombre. Una vez ingresado el nombre del archivo, el sistema reconoce las palabras que ya están agregadas al diccionario y las que no se encuentren en este, se requerirá que sean grabadas por el usuario. Luego de grabar cada palabra el sistema reproducirá la palabra, preguntará al usuario si la palabra se grabó correctamente y dará la posibilidad de regrabarla si asi lo desea el usuario.

```
root@osiris-desktop:/home/osiris/workspace/speakit# ant run
Buildfile: build.xml

init:

compile:

jar:

run:
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
1
     [java] Leer archivo de Texto
     [java]
     [java] Ingrese la ruta a continuación:
     [java] (Si su archivo es '1.txt' sólo presione ENTER)
```

```
root@osiris-desktop:/home/osiris/workspace/speakit# ant run
Buildfile: build.xml

init:

compile:

jar:

run:
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
1
     [java] Leer archivo de Texto
     [java]
     [java] Ingrese la ruta a continuación:
     [java] (Si su archivo es '1.txt' sólo presione ENTER)

     [java] El documento contiene palabras desconocidas, que deberá grabar a continuación.
     [java] Palabra 'esta'. (ENTER para grabar).
```

Si se ingresa un nombre de archivo que el sistema no puede encontrar, se emitirá un mensaje de error y la aplicación volvera al menu inicial.

```
jar:

run:
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
1
     [java] Leer archivo de Texto
     [java]
     [java] Ingrese la ruta a continuación:
     [java] (Si su archivo es '1.txt' sólo presione ENTER)
lalala.txt
     [java] No pudo encontrarse el archivo 'lalala.txt'.
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
```

- **Reproducir Archivo**: Con esta opción podemos reproducir las palabras grabadas con anterioridad en la aplicación. Se solicitará nuevamente el nombre del archivo ingresado y el sistema indicará por pantalla las palabras a reproducir consecutivamente y se escuchará su audio a continuación.

```
root@osiris-desktop:/home/osiris/workspace/speakit# ant run
Buildfile: build.xml

init:

compile:

jar:

run:
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
2
     [java] Ingrese la ruta a continuación:
     [java] (Si su archivo es '1.txt' sólo presione ENTER)

     [java] Reproduciendo: esta
     [java] Reproduciendo: es
     [java] Reproduciendo: una
     [java] Reproduciendo: prueba
     [java] Reproduciendo: de
     [java] Reproduciendo: speakit
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
```

Si se intenta reproducir un archivo, que contenga algunas palabras que no han sido grabadas con anterioridad el sistema lo indicara encerrando las palabras no grabadas entre corchetes y reproduciendo unicamente las que si se encuentran en el diccionario de audio.

```
     [java] Reproduciendo: que
     [java] Reproduciendo: [intentaban]
     [java] Reproduciendo: [explicar]
     [java] Reproduciendo: lo
     [java] Reproduciendo: que
     [java] Reproduciendo: [nos]
     [java] Reproduciendo: [rodea]
     [java] Reproduciendo: la
     [java] Reproduciendo: [ley]
     [java] Reproduciendo: de
     [java] Reproduciendo: [gravitación]
     [java] Reproduciendo: universal
     [java] Reproduciendo: [y]
     [java] Reproduciendo: la
     [java] Reproduciendo: [teoría]
     [java] Reproduciendo: [electromagnética]
     [java] Reproduciendo: clásica
     [java] Reproduciendo: [se]
     [java] Reproduciendo: [volvían]
     [java] Reproduciendo: [insuficientes]
     [java] Reproduciendo: [para]
     [java] Reproduciendo: [explicar]
     [java] Reproduciendo: [ciertos]
     [java] Reproduciendo: [fenómenos]
     [java] Speak It!
     [java] Menu Principal
     [java]     1.- Procesar archivo de Texto
     [java]     2.- Reproducir Archivo
     [java]
     [java]     0.- Salir
```

- **Salir**: Esta opción permite salir de **SpeakIt**.

Ejemplos

Junto a la distribución del sistema se incluyen unos documentos de ejemplos y los archivos binarios correspondientes (llamados AudioFile.dat y AudioFileIndex.dat). Para poder usar efectivamente estos ejemplos se deben colocar (junto a los binarios) dentro de la carpeta raiz del proyecto.
De estos documentos de ejemplos, el documento "ejemplo1.txt" ya ha sido grabado por nosotros, por lo que las palabras de este ya pertenecen al diccionario de audio. Los demás documentos quedan a disposición de ser utilizados en la evaluacion del trabajo práctico.
El detalle del contenido de los archivos de ejemplo es el siguiente:

ejemplo1.txt: *En física, la mecánica cuántica (conocida también como mecánica ondulatoria) es una de las ramas principales de la física que explica el comportamiento de la materia. Su campo de aplicación pretende ser universal, pero es en el mundo de lo pequeño donde sus predicciones divergen radicalmente de la llamada física clásica.*

ejemplo2.txt: *La mecánica cuántica es la última de las grandes ramas de la física. Comienza a principios del siglo XX, en el momento en que dos de las teorías que intentaban explicar lo que nos rodea, la ley de gravitación universal y la teoría electromagnética clásica, se volvían insuficientes para explicar ciertos fenómenos.*

ejemplo3.txt: *La teoría electromagnética generaba un problema cuando intentaba explicar la emisión de radiación de cualquier objeto en equilibrio, llamada radiación térmica, que es la que proviene de la vibración microscópica de las partículas que lo componen. Pues bien, usando las ecuaciones de la electrodinámica clásica, la energía que emitía esta radiación térmica daba infinito si se suman todas las frecuencias que emitía el objeto, con ilógico resultado para los físicos.*

ejemplo4.txt: *La mecánica cuántica rompe con cualquier paradigma de la física hasta ese momento, con ella se descubre que el mundo atómico no se comporta como esperaríamos. Los conceptos de incertidumbre, indeterminación o cuantización son introducidos por primera vez aquí.*

ejemplo5.txt: *Además la mecánica cuántica es la teoría científica que ha proporcionado las predicciones experimentales más exactas hasta el momento, a pesar de estar sujeta a las probabilidades.*

Los resultados esperados, al utilizar alguno de estos documentos teniendo en cuenta que las palabras del primer documento ya estan grabadas, serian los siguientes:

resultados para el ejemplo2.txt:
El sistema deberia pedir que se ingresen sólo las siguientes palabras: *última grandes comienza a principios del siglo XX momento dos teorías intentaban explicar nos rodea ley gravitación y teoría electromagnética se volvían insuficientes para ciertos fenómenos*

resultados para el ejemplo3.txt:
El sistema deberia pedir que se ingresen sólo las siguientes palabras: *teoría electromagnética generaba un problema cuando intentaba explicar emisión radiación cualquier objeto equilibrio térmica proviene vibración microscópica partículas componen pues bien usando ecuaciones electrodinámica energía emitía esta daba infinito si se suman todas frecuencias con ilógico resultado para los físicos*

resultados para el ejemplo4.txt:
El sistema deberia pedir que se ingresen sólo las siguientes palabras: *rompe con cualquier paradigma hasta ese momento se descubre atómico no comporta esperaríamos los conceptos incertidumbre indeterminación o cuantización son introducidos por primera vez aquí*

resultados para el ejemplo5.txt:
El sistema deberia pedir que se ingresen sólo las siguientes palabras: *además teoría científica ha proporcionado experimentales más exactas hasta momento a pesar estar sujeta probabilidades*

Obviamente, el sistema pedira ingresar estas palabras siempre y cuando el unico documento leido hasta el momento haya sido el primero, ya que si se lee alguno mas entonces las palabras que aparezcan en este no seran requeridas para ser ingresadas en futuros documentos.